These are release notes for ALN, the Atari Linker, detailing the changes between the FIRST release, version 1.00, on 6/26/87, and the latest release (dated 90/01/24).

SEARCH PATH FOR MODULES

If you have an environment variable "ALNPATH=path" then ALN will prepend "path" to file names when it can't find that file (or file.o) in the current directory. The command to set this might be

```
setenv ALNPATH=e:
```

The new command-line switch "-y path" does the same thing, and supercedes the ALNPATH environment variable if they are both present.

NEW COMMAND-LINE OPTIONS

All command-line switches may now be upper- or lower-case.

The -a option now has more flexibility: you can specify that any segment should come after any other. "xb" as a segment specifier means "put this bss," "xd" means "put this after data," and "xt" means "put this after text."

A new option, -h, allows you to set the hflags field of the PRG header in the linker output. "aln -h 7 ..." would set that field to 7 in the output. The value is entered the same way as -a arguments are. The meanings of the bits in this value are documented elsewhere.

A new option, -q, means "do a partial link (like -p), but also resolve commons into the BSS, so they're not common any more." Also, a new option, "-k name" means "add 'name' to a list of symbols; after the link, make all symbols EXCEPT those on the list local, not global." You may use -k for as many symbols as necessary. In combination with -q, this can add more modularity to a project: you might have two groups of object modules which need to share global symbols within the group, but not outside it. You can link each group into a single module using -q, then link these super-modules together to produce the program file.
Use -k for those symbols which really need to stay global: the symbols which are visible to the other super-modules.

A single -u causes unresolved symbols to be listed, but the link continues as if their values were zero. Another -u can now be used to suppress the listing, also.

BUG FIXES

The -d (desktop) flag didn't work as intended; now it does.

Link68 considers a symbol of type COMMON to be unresolved when it decides whether to extract a module from a library; now ALN does too.

ALN is now more compatible with the OLD Alcyon as68 (pre-version 4.14), especially for symbols of type COMMON: the old assembler generated a different type for these than the new one.

When linking absolute (with the -a flag), the symbol table was being written even if you didn't specify -s or -l. Now it is suppressed unless you specify one of those switches.

Various minor bug fixes for exceptional situations.

MISCELLANEOUS

Errors like 16-bit overflow used to cause the link to abort immediately. Now they are all reported, and THEN the link aborts.

ALN now clears the lower bit of the symbol type word as symbols are read, because it uses that bit internally. If your compiler produces "alcyon-compatible" object modules, but uses the lower bit of the symbol type word, you will lose that information. (The link should still work.)

ALN now supports extended argument processing on the MWC model. If your shell can use ARGV to pass arguments, you can pass an unlimited number of arguments to ALN. (The formal extended argument specification from Atari requires that the "length" byte of the command line be 0x7f to indicate that ARGV is in use, but this release of ALN does not insist on that because it is not yet widespread.)

Command files (included with the -c option) can now contain comments: a number-sign ("#") introduces a comment, and everything from there to the end of the line is ignored.

DOINDEX

Previously, doindex reported "incomplete file header in archive" for some perfectly normal archives; now it doesn't.

NAME

aln, doindex - GEMDOS linker and archive index builder

SYNOPSIS

aln [*option*] ...
{ *file* | [**-x** *file*] | [**-i** *file label*] } ...

doindex [**-i**] [**-w**] *file*

EXAMPLE

[in a batch file or from a command shell]
aln -o test -s -m gemstart test gemlib libf

This is a typical link for a program compiled with the C compiler: the startup file **gemstart.o** is linked first, followed by the object module produced by the compiler (test.o), plus the C run-time library **gemlib** and the floating-point library **libf**. This example produces an executable called test.prg (because of the **-o** flag), includes global symbols in that executable (the **-s** flag), and produces a load map on the standard output (the **-m** flag).

DESCRIPTION

File arguments are either object modules or archives of object modules (created with **ar68**). If *file* has no extension (e.g. '.o'), **aln** looks for both *file* and *file*.o. See the section FILE NAMES AND LIBRARY PATHS for more information.

All *options* must be specified before the first **-x**, **-i**, or *file* argument.

If no arguments are present on the command line, (i.e. **aln** was double-clicked from the GEM desktop) **aln** will display its version number and prompt for switches and filenames. More than one switch or filename may be entered per line, and a blank line ends the input and begins the link. After the link **aln** will wait for the user to press the return key before returning to the desktop (just like the -d flag).

The *options* are:

-a *text data bss*

Absolute link: *text*, *data*, and *bss* are indicators for each of those segments: a (hexadecimal) address, the letter 'r', or the letter 'x'. See the section on absolute linking, below.

-d

Desktop: wait for the "Return" key after the link before terminating. This gives the user time to read any error messages before

returning to the desktop. Note that if you start aln with no arguments, **-d** is implied.

-f

File symbols: when specified, **aln** will generate symbols of type "file" for each object module processed, and symbols of type "archive file" (bits 6 and 7 in the type field set) for each archive processed. These symbols cause **sid** to fail, but they can be understood by Atari's [newly-released] debugger, **db**. This flag sets the **-s** flag, unless **-l** is used also. See FILE SYMBOLS, below.

-l

Local symbols: like **-s**, but includes local symbols as well as globals in the output file.

-m

Map: produces a load map on standard output. The load map contains each symbol's name, value, and type. The load map lists only global symbols unless **-l** is used. The symbol types are encoded as follows:

C: Common	F: File
G: Global	A: Archive (only with "File")
E: External	Q: eQuated
L: Local	R: Register

-o *file*

Output: produce output on *file*. If *file* has an extension (e.g. '.prg'), that extension is used. Otherwise, a default extension is appended to *file*: '.prg' for a normal link, '.o' for a partial link (when **-p** is specified), and '.abs' for an absolute link (when **-a** is specified).

If **-o** is not specified, the output file name is taken from the first linked file on the command line (including archives specified with **-x** and data files specified with **-i**), plus the appropriate extension. Note that if this would make the output file name the same as the first input file (e.g. "aln -p a1.o a2.o" which would use "a1.o" as the output file name), **aln** will abort: in this case, **-o** must be specified.

-p

Partial link: collect the named object modules and libraries into a single object module, suitable for later passes through **aln**.

-s

Symbols: generate a symbol table in the output file including all global symbols. (Use **-l** (only) to get both locals and globals.)

-u

Unresolved: do not abort the link upon encountering unresolved externals. The unresolved symbols are listed on the standard output, but the link proceeds as if their values were zero.

-v

Verbose: causes **aln** to print a banner line at the start of the link and memory usage statistics at the end. If the -v flag is present twice, the name of each file (module or archive) is printed as it is linked. If this flag appears three times, **aln** also prints the name of each module it links from within each archive.

-w

Warnings: give warnings on standard output about multiply-defined globals. See the section DUPLICATE SYMBOLS IN MODULES for more information.

-y *pathname*

Library path: **Aln** will look in this directory as well as the current directory for object modules and archives. See the section FILE NAMES AND THE LIBRARY PATH for more information.

After the *options* on the command line, the following arguments are recognized in addition to *file* arguments:

-i *file label*

Include file: *file* will be included in the data segment verbatim. The global, data-segment symbol *label* is created and gets the starting address of *file* as its value; the global symbol *labelx* gets its ending address. That is, the ending label is the same as the starting label with the letter 'x' appended. If *label* is eight or more characters long, it is truncated to seven characters before appending the letter 'x' (so "longname" becomes "longnamx").

-x *file*

All modules: include all modules from archive *file*. Note that with the **-x** option, the modules are linked in the order they appear in the archive, so for multiply-defined global symbols, the first occurrence is the one which prevails. This is the opposite

behavior from the regular linking process.

The 'command-file' switch can appear anywhere:

-c *file*
Command file: *file* is read in as though its contents had appeared on the command line in place of the **-c** switch. All command-line switches may be used (but, again, no *options* may occur after the first **-i**, **-x**, or *file* argument in either the command line or the **-c** file). Arguments in *file* may be delimited by white space (tabs, spaces, and newlines) or commas.

DOINDEX -- ARCHIVES AND THEIR INDEXES

Aln requires that an index file exist for each archive included in a link. This index file has the same name as the archive, with the extension '.NDX', and should be in the same disk directory as the archive itself. If **aln** can not find an index file for an archive you name, it will produce an error message to that effect and abort.

Doindex builds an index file for the named archive (regardless of whether one already exists). If desired, **doindex** will also print a human-readable index of the archive on the terminal (that is, on standard output), and inform you of symbols which are declared global in more than one module in the archive. The **last** such declaration is the one which will prevail when that archive is used in the linking process.

The arguments to **doindex** are as follows:

-i
Index: print an index of the archive to the standard output, including the name of each module, the global symbols it exports, and the external symbols it imports. Finally, list the symbols which are external to the archive (imported by modules in the archive but not exported by any of them).

-w
Warnings: produce warnings about duplicate symbols in the archive.

The last argument to **doindex** is the name of an archive. **Doindex** opens that archive, builds its index file, and writes that file to *file*.ndx in the same disk directory as *file* itself.

The index file contains dependency information so the linker does not have to go through the whole archive to resolve all the symbols. It consists of information about each module in the archive, the name of each symbol exported by any module in the archive and the module which

exports it, and a dependency list for each module, stating, "if you need module A, you will also need modules B, C, and D." During linking, this information is collected together for each symbol which is unresolved at the time the archive appears in the command line, and only the needed modules are read in from the archive.

FILE NAMES AND THE LIBRARY PATH

Aln looks for files (modules and archives) both the current directory and in the directory named as the *library path*. The library path can be placed in the environment variable "ALNPATH" or named on the command line with the "-y" option. If both are present, the "-y" option supercedes the environment variable.

The library path should be a full pathname which names one directory, like "E:\LIB" -- including the drive letter, the colon, and the directory name (with leading backslash: anchored at the root). For the UNIX version, of course, the directory separator is the forward slash ("/"). It is necessary that the library path string contain at least one directory separator (backslash on the ATARI ST, slash on UNIX), so **aln** knows which one to place between the library path string and the file name.

When **aln** tries to open a file, it looks in a number of places for that file. First, it tries to open the file as the name appears, in the current directory. If that fails, **aln** appends ".o" to the file name and tries again in the current directory. Next, **aln** prepends the *library path* string to the name, and tries to open that file. Finally, **aln** prepends the library path *and* appends ".o" to the name. If none of these strategies works, **aln** gives up.

A file name can contain a partial path name: if you want to use the archive "E:\LIB\LOCAL\MYLIB," and your library path is "E:\LIB," then listing "LOCAL\MYLIB" on the command line is sufficient. (**Aln** will look first for "MYLIB" in the subdirectory "LOCAL" of the current directory, fail, and then prepend the library directory string, resulting in "E:\LIB\LOCAL\MYLIB," which is what you wanted.)

The above rules are not quite complete: **aln** never tries to append ".o" to a file name which already has a dot (".") in it. Also, **aln** does not prepend the library path string to names which already start with "/", or "\", or which contain a colon (":"). The assumption is that names starting with "/" or "\" are "anchored" at the root directory of the current drive, and that file names with colons in them refer to a different drive. These rules are more strict than they should be, perhaps, but they cover most situations. The "/" directory character is a concession to the UNIX(tm) version of **aln**; the other rules are inappropriate in that version, but do no harm.

ABSOLUTE LINKING

An **absolute link** is one for which the **-a** flag is specified. Note that the **-a** flag takes three arguments: the base indicator for the text,

data, and BSS segments, respectively. A base indicator can be one of the following:

A) a hexadecimal value, which is taken as the starting address of the segment.

B) the letter 'r', which stands for "relocatable."

C) the letter 'x', which stands for "contiguous with the previous segment" (whether that segment is absolute or relocatable).

During an absolute link, an absolute object module is produced, which includes the base address of each segment in its header. See the section on FILE FORMATS for more details.

In an absolute object module, all references to an absolute segment have already been resolved; that is, there is no relocation information for them, because they are not relocatable. References to relocatable segments still have relocation information associated with them. If there are no references to relocatable segments (either because there are no such segments, or no references to them), the relocation information is missing entirely, and a flag in the header indicates this.

For example, when linking a program to be placed in ROM, **aln** might be used to link with the text and data segments contiguous, starting at the address of the ROM (say, hex FF0000), and with the BSS segment at some address in RAM (say, hex 4000). This can be done with **aln** as follows:

```
aln -o rom.abs -a ff0000 x 4000 romfile.o
```

Alternatively, a program with its data segment in ROM, but with relocatable text and BSS segments, could be linked as follows:

```
aln -o romdata.abs -a r ff0000 r romfile.o
```

Of course, it would be up to the program loader to perform the text and BSS relocation at execute time.

FILE SYMBOLS

Aln will generate file symbols when the **-f** flag is used. A file symbol appears at the start of each object module in the symbol table. Its name is the name of the module, its value is the start of the text segment of that module, and its type is TEXT FILE (that is, hex 0280). With these symbols, you can determine which object module a given symbol came from, because the symbols from a module immediately follow its file symbol.

Aln also generates a file symbol at the start of each archive: this is a special symbol in that its name is the name of the archive, but its type is TEXT FILE ARCHIVE (hex 02C0). Furthermore, a second symbol is generated at the end of the archive: it has the same type, but its name is blank. This signals the end of the previous archive.

The use of bit 6 of the type field to mean "archive" is an extension to the Alcyon symbol-table standard. As such, existing tools like **sid** and **nm68** can not be expected to understand it, but Atari's [newly-released] debugger **db** does. (It happens that **sid** does not understand file symbols at all, let alone archive-file symbols, so the **-f** option should not be used if **sid** will be used to debug the output.)

FILE FORMATS

The files **aln** deals with all have the same basic format (except archives; see below):

Header
Text
Data
Symbol Table
Relocation Information

The header includes information such as the sizes of the other segments and the type of the file (encoded in a "magic number"). Any segment may be empty or missing except the header.

OBJECT MODULES

A standard (relocatable) object module header has the following format:

```
struct oheader {
        int magic;              /* the magic number 0x601a */
        long tsize;             /* text segment size */
        long dsize;             /* data segment size */
        long bsize;             /* bss segment size */
        long ssize;             /* size of the symbol table */
        char reserved[10];      /* ten unused bytes (must be zero) */
};
```

Following the header is the text segment of the module (tsize bytes long), then the data segment (dsize bytes long). Following that is the symbol table (ssize bytes long) and then the fixup information for the module. The fixup information matches word-for-word with the text, then data segments.

The fixup words have a type encoded in the lowest three bits:

VALUE	MEANING
0	absolute (needs no fixup)
1	data-segment relocatable

2	text-segment relocatable
3	bss-segment relocatable
4	symbol: see below.
5	marks the first word of a long fixup
6	pc-relative relocatable
7	marks the first word of an instruction

Fixup type 4 is a symbol-type fixup. The upper 13 bits of the fixup word are the index into the symbol table of the symbol being referenced. (Note that this means there may only be (2^13)-1 or 8191 symbols per module.)

EXECUTABLE PROGRAM FILES

Executable programs (.PRG files) have almost the same format as relocatable object modules. The header is the same, and the text and data segments, plus the symbol table, follow. The relocation information is different, though; it starts with a longword which is the byte offset to the first longword needing a fixup, then a series of bytes with the offset to the next longword needing a fixup. A zero terminates this list. "Fixing up" a longword means adding the base address of the text segment to it. There is no way to fix up a word.

If there are no fixups to be done, the initial-offset longword is zero. Note that this means the first longword of a .PRG file may not need fixing up; this is acceptable because that must be the first word of an instruction, and instructions never need fixing up. The byte-offsets take these special values:

Value	Meaning
0	End of fixup list.
1	Skip 254 bytes and keep going.
2..254 (even)	skip that many bytes and fix up.
3..255 (odd)	undefined.

ABSOLUTE OBJECT MODULES

An absolute object module header has the following format:

```
struct abshdr {
        int magic;        /* the magic number 0x601B */
        long tsize;       /* text segment size */
        long dsize;       /* data segment size */
        long bsize;       /* bss segment size */
        long ssize;       /* size of the symbol table */
        long reserved;    /* an unused longword */
        long textbase;    /* the base of the text segment */
        int relocflag;    /* zero if reloc info exists */
        long database;    /* the base of the data segment */
        long bssbase;     /* the base of the bss segment */
};
```


If there is any relocation information, the relocflag field in the header will be zero, and that information will follow the symbol table (if any). If the relocflag field is not zero (and in particular if it is minus one), there is no relocation information. This is always the case when none of the three segments is relocatable, but it can also happen if there are no references to a relocatable segment (e.g. the text segment is relocatable, but contains position-independent code, and the data and BSS segments are absolute).

ARCHIVES

Archives are files containing other files, usually relocatable object modules. The "header" of an archive file is simply the magic number FF65 (hex). The archived files consist of a header, then the file itself. The next file follows immediately. A zero word follows the last file in the archive. The archived-file header is as follows:

```
struct arheader {
        char a_fname[14];    /* the file name */
        long a_modtim;       /* the last-modified time */
        char a_userid;       /* not used in TOS */
        char a_gid;          /* not used in TOS */
        int a_fimode;        /* the file's mode word */
        long a_fsize;        /* the file's size in bytes */
        int reserved;        /* zero */
};
```

The file, a_fsize bytes' worth, immediately follows the header.

NOTES

DUPLICATE SYMBOLS IN MODULES

When the same symbol is exported (declared as global) from multiple object modules, the symbol value exported from the **first** such module will take precedence. When the same symbol is exported by multiple modules in one archive, the **last** such module will take precedence (this is taken care of by **doindex** when it builds the index). Therefore, in the case of two archives exporting the same symbol (from modules exporting needed symbols), the **last** definition in the **first** archive is the one which will be used.

However, if an archive is included with **-x**, the modules are read in archive order, and the **first** instance of a symbol is the one which prevails.

Unless the **-w** flag is used, you will get no notification that multiple files exported the same symbol.

UNUSED MODULES IN LIBRARIES

Since the dependency information is built from the archive, certain

conditions can cause it to be out-of-date with respect to a given link.

For example, if archive Z contains modules M and N, and M imports a symbol S and N exports it, then the index file for Z will reflect that M depends on N (that is, "if you need M you will also need N"). But if the symbol S is exported by a file Y earlier in a particular link, then module N is not actually needed at all.

Aln will read module N from the archive because of the dependency in the index file, but will then notice that both N and Y are exporting symbol S. This will produce a warning message if **-w** is specified. Since Y occurs earlier in the link than N, the value of symbol S is taken from Y. Finally, **aln** will notice that module N is not in fact used in the linking process, and will discard N completely, with another warning message.

EXAMPLES

(1) aln -o apskel apstart.o apskel.o gemlib aesbind vdibind

(2) aln -s -o myfile.prg -u t1 t2 gemlib
(3) aln -s -c foo.aln t1 t2 gemlib
(4) aln -s -o myfile.prg -c bar.aln

Example (1) links the GEM application apskel.o for execution.

Example (2) links t1.o and t2.o together, along with the library gemlib, to produce myfile.prg, with symbols.

Example (3) sets the **-s** flag, then reads foo.aln for more command-line options, and then specifies that t1 and t2 are files to be linked. If "foo.aln" contains "-o myfile.prg -u" then this is identical to example (2).

Example (4) sets **-s** and sets the output file to "myfile.prg," then reads bar.aln for more options. This time, if "bar.aln" contains "-u t1 t2 gemlib" then this is identical to examples (2) and (3).

ALN FOR LINK68 USERS

Aln performs the same function as **link68** plus **relmod.** If you have batch files which invoke **link68**, you can change them to use **aln** by doing the following:

1. Change bracketed options (like [u,s]) into the corresponding **aln** options (**-u -s**).

2. Change the assignment statement "output.68k=input1,input2,..." into something like "**-o** output.prg input1 input2 ..."

3. Change the command-file option (like [co[myfile.inp]]) into the corresponding **aln** option "**-c** myfile.inp"

4. Do the same things in any command files (like myfile.inp). Note that you don't have to remove the commas from your command files; **aln** will accept commas as separators in command files (but not on the command line).

For example, if you usually use the command lines:

```
link68 [s,u] myfile.68k=C:gemstart,myfile,C:gemlib
relmod myfile.68k myfile.prg
rm myfile.68k
```

you should change that to:

```
aln -s -u -o myfile.prg C:gemstart myfile C:gemlib
```

(From now on it will be assumed that you get rid of the **relmod** and **rm** commands, since **aln** generates ".prg" files directly.)

If you currently use a link68 input file, and your command line is like this:

```
link68 [s,u,co[myfile.inp]]
```

and the file "myfile.inp" contains a line like this:

```
myfile.68k=C:gemstart,myfile,C:gemlib
```

you should change that to:

```
aln -u -s -c myfile.inp
```

and "myfile.inp" should contain:

```
-o myfile.prg C:gemstart myfile C:gemlib
```

ERROR MESSAGES

Most of the common error messages from **aln** are self-explanatory; for instance, "File <x> is not an archive." In some cases, however, a little more explanation is in order.

Some errors refer to a 16-bit fixup overflow. This means that in resolving an external reference in the file, a value greater than 32K-1 or less than -32K had to be put in a single word. This can happen if you have a PC-relative reference to a symbol more than 32K away. This is only a warning, since you might be using the value as an unsigned integer (in which case it might not be an overflow).

Other errors report that they occurred at a given offset (always hex) in a given module. The offset is always in bytes, counting from the beginning of the text segment of that module.

At the end of the link, if the **-v** switch is present or **aln** was started with no arguments, you will see "Link complete." If there were any errors, you will see "Link aborted." In either case, some memory usage statistics are displayed as well.

AUTHOR

Allan K. Pratt, Atari Corp.

Atari can be reached electronically at atari!postmaster (on USENET), and Atari is available on the online services GEnie, BIX, and Compuserve.

